Efficient Snapshot Retrieval over Historical Graph Data

Udayan Khurana, Amol Deshpande
University of Maryland, College Park
{udayan, amol}@cs.umd.edu
Agenda

• What
  – Historical Graph Data
  – Snapshot Retrieval

• Why
  – Motivation

• How
  – GraphPool
  – DeltaGraph

• Experiments
Historical Graph Data

• A collection of graph snapshots
  – A set of nodes and a set of edges
    • A list of attribute-value pairs

• Event
  – Creation or deletion of an edge or node
  – Change in an attribute value of a node or an edge

(a) \{NE, N:23, N:4590, directed:no, 11/29/03 10:10\}
(b) \{UNA, N:23, ‘job’, old:‘..’, new:‘..’, 11/29/07 17:00\}

\[ G_k = G_{k-1} + E, \quad G_{k-1} = G_k - E \]
Snapshot Retrieval

- **Singlepoint**
  - `GetHistGraph(Time t, String attr options)`

- **Multipoint**
  - `GetHistGraphs(List<Time> t list, String attr options)`
  - `TimeExpression`
  - `Time interval`

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-node:all</td>
<td>None of the node attributes</td>
</tr>
<tr>
<td>+node:all</td>
<td>All node attributes</td>
</tr>
<tr>
<td>+node:attr1</td>
<td>Node attribute named “attr1”; overrides “-node:all” for that attribute</td>
</tr>
<tr>
<td>-node:attr1</td>
<td>Node attribute named “attr1”; overrides “+node:all” for that attribute</td>
</tr>
</tbody>
</table>
/* Loading the index */
GraphManager gm = new GraphManager(...);
gm.loadDeltaGraphIndex(...);
...

/* Retrieve the historical graph structure along with node names as of Jan 2, 1985 */
HistGraph h1 = gm.GetHistGraph("1/2/1985", "+node:name");
...

/* Traversing the graph*/
List<HistNode> nodes = h1.getNodes();
List<HistNode> neighborList = nodes.get(0).getNeighbors();
HistEdge ed = h1.getEdgeObj(nodes.get(0), neighborList.get(0));
...

/* Retrieve the historical graph structure alone on Jan 2, 1986 and Jan 2, 1987 */
listOfDates.add("1/2/1986");
listOfDates.add("1/2/1987");
List<HistGraph> h1 = gm.getHistGraphs(listOfDates, "");
...
Motivation

• To support a broad range of network analysis tasks
• The existing solutions for graph data management lack adequate techniques for temporal annotation
System Architecture

- **Social Network Analysis Software**
  - Analyst
  - QueryManager: Translate user query into Graph Retrieval and execute Algorithms on graphs

- **System**
  - GraphManager: Manage GraphPool - Overlaying historical graphs and cleanup
  - GraphPool: Active Graph Pool Table (Query, Time, Bit, Graph)
  - HistoryManager: Manage DeltaGraph - Query Planning, Disk Read/Write
  - Key-Value Store: DeltaGraph
GraphPool

• An in-memory graph data structure in an overlapping manner
• A single graph that is the union of
  – current graph
  – historical snapshots
  – materialized graphs
• GraphID-Bit Mapping Table
• Update and Clean-up
GraphPool

GraphPool(current, t1, t2)

GraphId-Bit Mapping Table

<table>
<thead>
<tr>
<th>Bit</th>
<th>GraphID</th>
<th>Graph</th>
<th>Dep</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,3</td>
<td>34</td>
<td>Hist. Graph</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Mat. Graph</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
<td>Mat. Graph</td>
<td>-</td>
</tr>
<tr>
<td>6,7</td>
<td>35</td>
<td>Hist. Graph</td>
<td>4</td>
</tr>
</tbody>
</table>
DeltaGraph

- A novel, extensible, highly tunable, and distributed hierarchical index structure
- Prior Techniques
  - External interval tree, Segment trees, Snapshot index, Copy+Log
  - Not efficiently support multipoint queries, Not highly tunable, Not support different persistent storage options
DeltaGraph

(a)

Super-Root

$S_8 = \emptyset$

$\Delta(S_7, S_8)$

Root

$S_7 = f(S_5, S_6)$

$\Delta(S_5, S_7)$

$\Delta(S_6, S_7)$

$S_5 = f(S_1, S_2)$

$\Delta(S_1, S_5)$

$\Delta(S_2, S_5)$

$S_6 = f(S_3, S_4)$

$\Delta(S_3, S_6)$

$\Delta(S_4, S_6)$

$S_1$

$S_2$

$S_3$

$S_4$

$L$

$E_1$

$E_2$

$E_3$
DeltaGraph

(a) Singlepoint query $t_1$

(b) Multipoint query $\{t_1, t_2, t_3\}$
DeltaGraph

- Model of Graph Dynamics

\[ |G_{|E|} = |G_0| + |E| \times \delta_* - |E| \times \rho_* \]

- Differential Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersection</td>
<td>( f(a, b, c \ldots) = a \cap b \cap c \ldots )</td>
</tr>
<tr>
<td>Union</td>
<td>( f(a, b, c \ldots) = a \cup b \cup c \ldots )</td>
</tr>
<tr>
<td>Skewed</td>
<td>( f(a, b) = a + r.(b - a), 0 \leq r \leq 1 )</td>
</tr>
<tr>
<td>Right Skewed</td>
<td>( f(a, b) = a \cap b + r.(b - a \cap b), 0 \leq r \leq 1 )</td>
</tr>
<tr>
<td>Left Skewed</td>
<td>( f(a, b) = a \cap b + r.(a - a \cap b), 0 \leq r \leq 1 )</td>
</tr>
<tr>
<td>Mixed</td>
<td>( f(a, b, c \ldots) = a + r_1.(\delta_{ab} + \delta_{bc} \ldots) - r_2.(\rho_{ab} + \rho_{bc} \ldots), 0 \leq r_2 \leq r_1 \leq 1 )</td>
</tr>
<tr>
<td>Balanced</td>
<td>( f(a, b, c \ldots) = a + \frac{1}{2}.(\delta_{ab} + \delta_{bc} \ldots) - \frac{1}{2}.(\rho_{ab} + \rho_{bc} \ldots) )</td>
</tr>
<tr>
<td>Empty</td>
<td>( f(a, b, c \ldots) = \emptyset )</td>
</tr>
</tbody>
</table>
Experiments

• Implementation
  – Java
  – Kyoto Cabinet key-value store

• Datasets
  – DBLP
  – Randomly generated small
  – Randomly generated large
Experiments
Experiments

(a) Memory (MB) vs. Query Count
- Dataset 2
- Dataset 1

(b) Avg Graph Retrieval Time (ms) vs. # Cores
- Avg Query Time

(c) Graph Retrieval Time (ms) vs. # Queries
- SinglePoint Queries
- Multipoint Query

(d) Graph Retrieval Time (ms) vs. Query Timepoint
- Structure+Attributes
- Structure Only
Experiments

(a) Varying Arity (Dataset 1)

(b) Varying EventList Size (Dataset 1)

(a) Average Query Time

(b) Materialization Memory Requirement
Experiments

(a) Graph Retrieval Time (ms) vs Query Timepoint
(b) Graph Retrieval Time (ms) vs Query Timepoint

Legend:
- Balanced
- Intersection
- Balanced (root materialized)

Lines:
- r1=0.1, r2=0.1
- r1=0.5, r2=0.5
- r1=0.9, r2=0.9
Thank You!